DISPLAY OF MENU ITEMS IN A USER INTERFACE

FIELD OF THE INVENTION

The present invention relates to a method for displaying menu items in a user interface and in particular to such a method for use with a device for use with a mobile communications network.

BACKGROUND OF THE INVENTION

One of the limitations that is common in many mobile devices is that the display screen is quite small and when a menu is displayed it is not always possible to display all of the menu items on the screen at one time. Conventional approaches tend to load all of the menu items into memory, along with associated icons or graphics, and then display them appropriately as the user scrolls up or down the menu. This approach is not an efficient technique for devices that are resource limited, such as mobile telephones, and for devices that use a mark-up language to render the device display and/or to provide the operating software for the device.

SUMMARY OF THE INVENTION

According to a first aspect of the present invention, there is provided a method of displaying a subset of a plurality of user interface elements in a user interface, the method comprising the steps of: (i) determining the size of the subset of plurality of UI elements that can be displayed within the user interface; (ii) determining a plurality of UI elements that may be selected for display within the user interface; (iii) selecting the subset of UI elements from the plurality of UI elements determined in step (ii); and (iv) displaying the subset of UI elements selected in step (iii) within the user interface. Step (iii) may be repeated to select a further subset of UI elements in response to a user input and step (iv) is then repeated to display the further subset of UI elements within the user interface.

According to a second aspect of the present invention, there is provided a data carrier comprising computer executable code for performing any of the above methods.

According to a third aspect of the present invention, there is provided a device comprising a display and a user interface the device being configured, in use, to (i) determine the size of a subset of plurality of UI elements that can be displayed within the user interface; (ii) determine a plurality of UI elements that may be selected for display within the user interface; (iii) selecting the subset of UI elements from the plurality of UI elements determined in step (ii); and (iv) displaying the subset of UI elements selected in step (iii) within the user interface.

According to a fourth aspect of the present invention, there is provided a device comprising processing means, storage means, a display, user input means, wireless communication means and a user interface, wherein the device is configured to perform the method of any of the above described methods.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a schematic depiction of a system incorporating the present invention;
Figure 2 depicts in greater detail the structure and operation of server;
Figure 3 shows a schematic depiction of the software for the mobile devices; and
Figure 4 shows a schematic depiction of a device that comprises a user interface according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention will now be described by way of illustration only and with respect to the accompanying drawings, in which Figure 1 shows a schematic depiction of a system incorporating the present invention. The system comprises server 100, content toolset 200, mobile devices 300, operational support systems (OSSs) 700, content feeds 500 and user interface (UI) sources 600. In use, the server 100 communicates content data and UI data to the mobile devices 300, 301, …, each of which comprise software package 400. The server 100 interfaces with OSSs 700, with the OSSs being those conventionally used to operate mobile networks, for example billing, account management, etc. The server 100 further interfaces with the content toolset 200: the content toolset receives data from UI sources 600, 601, …, and packages the UI data such that the server can transmit the packaged UI data to the software packages 400 comprised within the mobile devices 300. The server receives data from a plurality of content feeds, and this data is processed and packaged such that it can be sent to the software packages 400 or so that the mobile devices 300 can access the data using the software package 400.

The system can be envisaged as being divided into three separate domains: operator domain 50 comprises the systems and equipment operated by the mobile network operator (MNO); user domain 60 comprises a plurality of mobile devices and third-party domain 70 comprises the content feeds and UI feeds that may be controlled or operated by a number of different entities.

Figure 2 depicts in greater detail the structure and operation of server 100. Server 100 comprises publishing component 110 and content server component 150. Publishing component comprises database 111, import queue 112, content toolset interface 113, user interface 114 & catalogue 115. In operation, the publishing component receives content from the content toolset at the content toolset interface. The content is presented in the form of a parcel 210a, 210b, …, (see below) comprising one or more trigs and one or more triglets. A trig is a user interface for a mobile device, such as a mobile telephone and a triglet is a data file that can be used to extend or alter a trig. If a parcel comprises more than one trig then one of the trigs may be a master trig from which the other trigs are derived.

Figure 3 shows a schematic depiction of the software 400 for the mobile devices 300, which comprises a mark-up language renderer 410, update manager 420, network communication agent 425, resource manager 430, virtual file system 435, actor manager 440, a plurality of actors 445a, 445, …, native UI renderer 450, support manager 460, trig manager 465 and mark-up language parser 470.

It is preferred that the software operates using TrigML, which is an XML application and that mark-up language renderer 410 renders the TrigXML code for display on the mobile device 300. The mark-up language renderer also uses the TrigML Parser to parse TrigML resources, display content on the device screen and control the replacement and viewing of content on the handset. The native UI renderer is used to display UI components that can be displayed without the use of TrigML, and for displaying error messages.

The software 400 is provisioned and installed in a device specific manner. For example for a Nokia Series 60 device the software is installed using a SIS file, whereas for a MS Smartphone device the software is installed using a CAB file. Similarly, software upgrades are handled in a device specific manner and upgrades may be provided over the air. The software may be provisioned in a more limited format for example as a self-contained application that renders its built in content only, that is additional trigs cannot be added later to the trig that is supplied with the application.

The trig manager 465 presents an interface to the resource manager 430 and the mark-up language renderer. It is responsible for trig management in general. This includes: persisting knowledge of the trig in use, changing the current trig, selection of a trig on start-up, selection of a further trig as a fall back for a corrupt trig, maintaining the set of installed trigs, identifying where a particular trig is installed to the resource manager and reading the update channel definitions of a trig and configuring the update manager appropriately.

The resource manager provides an abstraction of the persistent store on device, i.e. storing the files as real files, or as records in a database. The resource manager presents a file system interface to the mark-up language renderer and the update manager. It is responsible for handling file path logic, distinguishing between real resource files and actor attributes, mapping trig-relative paths onto absolute paths, interfacing with the trig manager and providing a modification interface to the update manager.

The Resource Manager is also responsible for ensuring the integrity of the resources stored in the persistent store, especially in the face of unpredictable interruptions such as loss of device power. The Resource Manager has no knowledge of the trig currently used. Its interface is thread safe (as it may be used by both the Update Manager and the Renderer from different threads.

The Update Manager handles the reception and application of trigs and triglets. The Update Manager presents an interface to the Renderer and the trig Manager and is responsible for: the initiation of manual updates when instructed to by the Renderer; controlling and implementing the automatic update channel when so configured by the trig manager; indicating the progress of a manual update and recovering an Update following unexpected loss of network connection and/or device power.. The update packet format may be defined as a binary serialisation of an XML schema.

The Support Manager provides an interface for other components to report the occurrence of an event or error. Depending on the severity of the error, the Support Manager will log the event and/or put up an error message popup

The Actor Manager 440 looks after the set of actors 445 present in the software. It is used by: the renderer when content is sending events to an actor; actors that want to notify that an attribute value has changed and actors that want to emit an event (see below).

The software may comprise a multi-threaded application running a minimum of two threads, with more possible depending on how many and what sort of actors are included. The software runs mostly in one thread, referred to as the main thread. The main thread is used to run the renderer which communicates synchronously with other components. Actors always have a synchronous interface to the Renderer. If an actor requires additional threads for its functionality, then it is the responsibility of the Actor to manage the inter-thread communication. It is preferred that a light messaging framework is used to avoid unnecessary code duplication where many actors require inter-thread communication.

In addition to the main thread, the update manager runs a network thread. The network thread is used to download update packets and is separate from the main thread to allow the renderer to continue unaffected until the packet has arrived. The Update Manager is responsible for handling inter-thread messaging such that the Update Manager communicates synchronously with the Renderer and Resource Manager when applying the changes defined in an Update Packet.

The memory allocation strategy of the software is platform specific. On MIDP platforms, the software simply uses the system heap and garbage collector for all its memory requirements. Garbage collection is forced whenever a content replacement event occurs in an attempt to keep the garbage collection predictable and not suffer unexpected pauses in operation. It is assumed that any memory allocation might fail, in which case the software will delete all its references to objects, garbage collect, and restart - assuming that the software has already successfully started up and rendered the first page.

On C++-based platforms, a mixture of pre-allocation and on-demand allocation will be made from the system heap. All memory required for start-up is allocated on-demand during start-up, with any failures here causing the exit (with message if possible) of the software. Following successful start-up, memory needed for rendering the content document model is pre-allocated. Provided content is authored to use less than a defined limit, it is guaranteed to render. Additional use is made of RAM for various caches needed for fast operation of the software. Where memory conditions are low, these caches will be released resulting in slow rendering performance from the software.

The Renderer receives information regarding the key press. If there is no behaviour configured at build time for a key, it is sent as a TrigML content event to the current focus element. The content event is then handled as defined by TrigML's normal event processing logic.

For example, if a key is pressed down, a 'keypress' event is delivered to the Renderer with a parameter set to the relevant key. When the key is released, a '!keypress' event is delivered to the Renderer. If a key is held down for a extended period of time, a 'longkeypress' event is delivered to the renderer. On release, both a '!longkeypress' and a '!keypress' event are delivered to the Renderer.

Whenever the software is started, it executes the following actions:

- Check for, and continue with, interrupted Update processing;
- Check for, and process, Updates residing in the file system (either pre-provisioned, or installed to the file system by some other means);
- If known, start the current trig (which may be the last run trig);
- If a current trig is not set, a trig that has been flagged as a 'default' trig can be started.
- Failing the presence of a default trig, the first valid trig by alphabetical order of name will be selected.

A trig is started by loading the defined resource name, start-up/default. The TrigML defined in start-up/default is parsed as the new contents for the content root node.

The first time a trig is run by the software following its installation, the trig is started by loading the resource name startup/firsttime. The software may record whether a trig has been run or not in a file located in the top level folder for that trig. Dependent on the platform used by the mobile device, the automatic start-up of the software may be set as a build-time configuration option. Furthermore, placing the software in the background following an auto-start may also be a build-time configuration option.

A launcher may appear to the user as an application icon and selecting it starts the software with a trig specified by that launcher (this trig may be indicated by a launcher icon and/or name). When using a launcher to start a trig, it is possible to specify an 'entry point' parameter. The parameter is a resource name of a file found in the 'start-up' folder. This file is not used if the trig has never been run before, in which case the file called 'firsttime' is used instead.

The software uses content resource files stored in a virtual file system on the device. The file system is described as virtual as it may not be implemented as a classical file-system, however, all references to resources are file paths as if stored in a hierarchical system of folders and files. Furthermore, the software stores some or all of the following information: usage statistics; active user counts; TrigManager state; TrigML fragments & update channel definition (serialised as binary XML); PNG images; plain text, encoded as UTF-8 OTA and then stored in a platform specific encoding; other platform specific resources, e.g. ring tone files, background images, etc.

Files in the file system can be changed, either when an actor attribute value changes, or when a file is replaced by a triglet. When files in the /attrs directory change, the

Renderer is immediately notified and the relevant branches of the content tree are updated and refreshed. When images and text resources are changed, the Renderer behaves as if the affected resources are immediately reloaded (either the whole content tree or just the affected branches may be refreshed). When TrigML fragments are changed, the Renderer behaves as if it is not notified and continues to display its current, possibly out of date, content. This is to avoid the software needing to persist <include> elements and the <load> history of the current content.

The software 400 is provisioned to mobile devices in a device specific method. One or more trigs can be provisioned as part of the installation, for example, stored as an uncompressed update packet. On start-up, the packet can be expanded and installed to the file-system.

The Actors 445 are components that publish attribute values and handle and emit events. Actors communicate with the Renderer synchronously. If an actor needs asynchronous behaviour, then it is the responsibility of the actor to manage and communicate with a thread external to the main thread of the Renderer.

One of the limitations that is common in many mobile devices is that the display screen is quite small and when a menu is displayed it is not always possible to display all of the menu items on the screen at one time. Conventional approaches tend to load all of the menu items into memory, along with associated icons or graphics, and then display them appropriately as the user scrolls up or down the menu. This approach is not an efficient technique for devices that are resource limited, such as mobile telephones, and for devices that use a mark-up language to render the device display and/or to provide the operating software for the device.

This problems is addressed by providing an efficient technique that enables a limited number of menu items to be loaded into memory, such that these menu items can be displayed simultaneously by the device. When the user scrolls up or down the menu, the item(s) no longer on display are discarded and the item(s) now on display are loaded into memory.

Preferably, this can be implemented by using a <griddata> element in TrigML to define a list view of some data, where the data is stored in a folder in the file system, and the list appearance has the same structure for each item. The <griddata> element comprises a 'repeat-over' attribute that specifies the folder in which the data can be located. The single child element of <griddata> is a template for the appearance of each item in the list.

The template may use a special symbol, e.g. '$$' to refer to the iterator. This is the template variable that changes each time the template is instantiated: for example

```
<griddata repeatover="news/headlines">
     <text res="news/headlines/$$/title.txt"/>
</griddata>
```

where the folder news/headlines/ contains:

```
     0/title.txt
```

```
1/title.txt
2/title.txt
3/title.txt
```

This would display a list of 4 items, each described by a simple <text> element pointing the 'title.txt' resource in the 'news/headlines/$$' folder. Where the source data has more items in it than the <griddata> element has room for on the display, the <griddata> element only displays those items that can be displayed. When the user scrolls through the list, the <griddata> element shifts the 'data-window' accordingly. The advantage of this technique is that only the resources required by the current display are actually loaded in memory, which reduces the memory utilisation and reduces the amount of time taken to render the list of items.

A similar scheme can be used to define the order that a list is displayed in. If the target of the 'repeat-over' attribute is a file instead of a folder, then the file can be assumed to contain a list of resource names to use in the iteration. For example,

```
<griddata repeatover="football/league">
     <text res="football/teams/$$/name.txt"/>
</griddata>
```

where the file football/league contains:

```
Manchester
Arsenal
Chelsea
```

the folder football/teams/ contains:

```
Manchester/name.txt
```

```
Arsenal/name.txt
Chelsea/name.txt
```

and each name.txt is a text file holding the team name. The result of this is that the test files associated with the teams would be displayed in the defined order and within the defined area of the device display.

A further example of this is given below, with the template causing a list of surname,firstname data to be displayed, with the list comprising three entries.

```
<gridlist id="myList" repeatover="path/to/list" rows="3">
  <group>
    <text res="$$/surname"/>
    <text res="$$/firstname"/>
  </group>
</gridlist>
```

This approach can be extended to enable the lists to be nested within other lists. Following the above approach, if there are a number of different lists, then there is ambiguity over which list the '$$' operator refers to. This problem is addressed by replacing '$$' with an expression that identifies uniquely the list it is referring to, such as for example:

```
{/elem/myList/index}
```

In addition to being able to uniquely refer to a list, this approach enables items to reference the list but not actually be in the list. For example, the title bar of a window could contain text relating to the selected item in a list and the text in the title bar could change as the list was scrolled through.

Where data is accessed by means other than the file system, e.g. it is stored in a database, or it is generated on the fly by another software component, this scheme can still be used if a virtual file system 435 is used, which can map a file system interface onto the underlying provider of the data. This means the content can still be arranged as described above, but the data can be provided in a method that enables efficient data storage and retrieval.

Updates sent OTA can be fetched using HTTP-GET requests initiated by the software. The GET request is directed at the URL associated with the Update. The body of the HTTP response is a binary file carrying data in an Update Packet format. The data reception is handled in a separate thread to the Renderer thread. For background updates (automatically initiated) this allows the user to continue navigating the UI. For foreground updates (manually initiated) this allows the Renderer thread to display a progress bar and listen for a cancel instruction.

The algorithm used to unpack and install an update is device specific. However, it is important that the algorithm is safe from unexpected interruption (e.g. power loss), such that no corruption or unrecoverable state is reached in the file-system. This may be achieved by using two threads (a network thread and a renderer thread) with the goal of having as much of the update processing as possible being performed by the network thread so as to interrupt the renderer thread for the shortest possible amount of time.

There are other failure modes to consider: if an HTTP-GET cannot be initiated, or is met with an HTTP error response code, then this attempt at an Update is abandoned and the retry strategy is used to begin a new update attempt at a later date. Where an HTTP response is interrupted by loss of network signal, any temporary files are deleted and the retry strategy is used to restart the Update attempt at a later date. If an update header indicates that the update payload size may be too great to fit on the device, if the update requires an incompatible version of the software or if the Update already resides on the device then the header data file is deleted and the Update attempt and any subsequent retries are cancelled.

TrigML fragments are files containing text TrigML and resource references inside these fragments are virtual file paths. The mapping of these virtual file paths to real file paths is defined by a TrigDefinition file. This file also defines other properties of the trig. When used for compiling a triglet, this file also defines how the input TrigML/PNG/Text resources map onto modifications of the virtual file-system of a trig.

In order to successfully render the user interface of a mobile device, the mark-up language must have the following qualities: concise page definitions, consistent layout rules, be implementable in a compact renderer, provide multiple layering and arbitrary overlapping content, event model, require the repaint of only the areas of the display that have to change between pages of the UI, include hooks to the platform for reading property values receiving events and sending events, extensible, and be graphically flexible. TrigML provides these features and an overview of the elements and attributes that provide the desired functionality can be found in our co-pending application GB0403709.9, filed February 19th 2004.

It is desirable that the cost of re-branding UIs and producing a continual stream of updates is minimal. This is enabled by providing an efficient flow of information from the creative process through to the transmission of data to users. A container, referred to as a parcel, is used for UIs, UI updates, and templates for 3rd party involvement. Parcels contain all the information necessary for a 3rd party to produce, test and deliver branded UIs and updates.

Many different UIs can be derived from a common base. Typically the common base would implement most of the interface itself, and trigs derived from it would implement small variations on it, such as branding. A Triglet can be derived from a Trig, and it can override any of the resources from the parent Trig that it chooses to (optionally it may introduce its own resources). Note that "resources" here also refers to TrigML, so the behaviour and layout of a Trig can be modified by a Triglet just as easily as it replacing a single image or piece of text.

A Parcel may comprise one or more base trigs (i.e. a Trig that is not derived from any other trig), one or more multiple trigs derived from a base Trig, a plurality of triglets derived from any of the trigs and a plurality of triglets derived from other triglets.

The parcel format is an opaque binary format that stores all this information as serialized objects. The parcel may comprise a number of resources , such as images, text, URLs, update channels, ringtone files, wallpapers, native applications, etc. Each resource contains permission information as to how to view, edit, or delete the resource. The parcels may be used to develop trigs and/or triglets for mobile devices having different capabilities such as display size, RAM capacity. To simplify this, a number of hierarchies may be defined and the data resource or TrigML element classified within the hierarchies. When a trig or triglet is compiled from a parcel, the most appropriate resources or TrigML elements can be selected and complied for a particular device.

Figure 4 shows a schematic depiction of a device 800 that comprises a user interface according to an embodiment of the present invention. The device comprises a display 810 that displays the user interface 815 and user interface means 820, that enable the user to interact with the user interface 815. A processor 830 executes the software that is stored within one or more storage means 840 and there may be provided one or more wireless communication interfaces 850, to enable communication with other devices and/or communication networks. One or more batteries 860 may be received to power the device, which may also comprise interfaces to receive electrical power and/or communication cables.

The nature of these components and interfaces will depend upon the nature of the device. It will be understood that such a user interface can be implemented within a mobile or cellular telephone handset, but it is also applicable to other portable devices such as digital cameras, personal digital organisers, digital music players, GPS navigators, portable gaming consoles, etc. Furthermore, it is also applicable to other devices that comprise a user interface, such as laptop or desktop computers.

The user interface means may comprise a plurality of buttons, such as a numerical or alpha-numerical keyboard, or a touch screen or similar. One or more storage devices may comprise a form of non-volatile memory, such as a memory card, so that the stored data is not lost if power is lost. ROM storage means may be provided to store data which does not need updating or changing. Some RAM may be provided for temporary storage as the faster response times support the caching of frequently accessed data. The device may also accept user removable memory cards and optionally hard disk drives may be used as a storage means. The storage means used will be determined by balancing the different requirements of device size, power consumption, the volume of storage required, etc.

Such a device may be implemented in conjunction with virtually any wireless communications network, for example second generation digital mobile telephone networks (i.e. GSM, D-AMPS), so-called 2.5G networks (i.e. GPRS, HSCSD, EDGE), third generation WCDMA or CDMA-2000 networks and improvements to and derivatives of these and similar networks. Within buildings and campuses other technologies such as Bluetooth, IrDa or wireless LANs (whether based on radio or optical systems) may also be used. USB and/or FireWire connectivity may be supplied for data synchronisation with other devices and/or for battery charging.

Computer software for implementing the methods and/or for configuring a device as described above may be provided on data carriers such as floppy disks, CD-ROMS. DVDs, non-volatile memory cards, etc.

This application claims the benefit of UK Patent Application number 0403709.9, filed February 19th 2004, the contents of which are incorporated herein by reference.